A FAST EXPECTED TIME ALGORITHM FOR THE POINT PATTERN MATCHING PROBLEM

P.B. VAN WAMELEN, Z. LI, AND S.S. IYENGAR

Dedicated to Professor R.L.Kashyap on the occasion of his 61st birthday (Purdue University).

ABSTRACT. Point set pattern matching is an integral part of many pattern recognition problems. The intent of this paper is the design and analysis of a probabilistic similarity transformation matching algorithm.

If n is the number of points in the patterns to be matched, we give a $O(n(\log n)^{3/2})$ expected time algorithm for the point pattern matching problem and show that it is faster than any existing algorithms in the literature. We then describe some experimental results on both fingerprints and randomly generated data for the validation of our theoretical analysis. These results show significant improvements in running time. Our theoretical analysis is heuristic in nature but seems to agree well with the experimental results.

1. INTRODUCTION

The design and analysis of data structures and algorithms is an important area of point pattern matching algorithms. In recent years there have been several significant advances in this area, ranging from $O(n^6)$ complexity to $O(n^2)$ algorithms. These advances have focused on developing faster algorithms and furthermore the results have kindled a lot of interest in obtaining an optimal algorithm in this area. These results have produced efficient solutions to many real-life problems. For details see [10], [7], [11], [3], [1], [2]. A comparison of these results are reviewed in Table 1.

2. STATEMENT OF THE PROBLEM

Suppose two point patterns P and Q in two dimensions are given. That is $P = \{p_1, p_2, \ldots, p_n\}$ and $Q = \{q_1, q_2, \ldots, q_m\}$, where the p_i and q_j are points in $\mathbb{R}^2$. We will assume that P and Q have approximately the same number of points. We want to find a similarity transformation T_{s,θ,t_x,t_y}, such that $T(P)$ "matches" Q, where matching will be made precise below. In the transformation T_{s,θ,t_x,t_y}, s is a scaling factor, θ a rotation angle and t_x and t_y the x and y translations respectively. That is, for $(x, y) \in \mathbb{R}^2$, we have

$$T\begin{pmatrix} x \\ y \end{pmatrix} = \begin{pmatrix} t_x \\ t_y \end{pmatrix} + s\begin{pmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{pmatrix}\begin{pmatrix} x \\ y \end{pmatrix}.$$

To define a match we assume two parameters, the matching probability, $\rho \in [0, 1]$ and the matching size, $t \in \mathbb{R}^+$ are given. Then we say $T_{s,\theta,t_x,t_y}(P)$ matches Q if there exists a subset $P' \subset P$ with at least ρn elements such that for each $p \in P'$ we have $|T_{s,\theta,t_x,t_y}(p) - q| < t$ for some $q \in Q$.

What is a reasonable value for t? Suppose the radius of the set of points Q is r (that is, the point set Q lies in a disk of radius r). Then, assuming uniform distribution of points, the average

Date: November 14, 2000.

Key words and phrases. point pattern matching, fast algorithm.

S.S. Iyengar was partially supported by a DOE-ORNL grant.

TABLE 1. History of results on point set pattern matching algorithms.

Year	Researcher	Technique	Geometric Properties	Complexity[1]
1980	Ranade and Rosenfeld [10]	Relaxation Approach	Translation differences	$O(n^4)$
1984	Ogawa [7]	Fuzzy Relaxation	Translation, rotation, scaling differences	$O(n^6)$
1993	Vinod and Ghose [11]	Asymmetric Neural Networks	Translation, rotation, distortion and noise but not scaling	
1997	Chang et al [3]	2D Cluster Approach	Translation, rotation, scale changes, local distortion, extra and or missing points	$O(n^4)$
1998	Boxer [1]	A sequential algorithm	Translations and Rotation differences in 3D	$O(n^2(\lambda_6(n)/n)^{1/2})\log n)$
1998	Chang et al [2]	Nearest Neighbors Search	Translation, rotation, scaling, local distortion and extra/missing points	$O(k^2n^2)$
1999	van Wamelen et al	Probabilistic, Sorted Nearest Neighbors	Translation, rotation, scaling, local distortion and extra/missing points	$O(n(\log n)^{3/2})$

shortest distance between points in Q is $r/(2\sqrt{n})$, see Section 4.1, Equation 4. It seems reasonable to choose t to be some constant fraction of this distance. Let λ be this fraction, we will call it the matching factor. For the rest of the paper we will assume that t was picked in this way, that is,

$$t = \lambda \frac{r}{2\sqrt{n}}. \tag{1}$$

Note that in general when we think of larger point sets we think of them as also growing in size. That is, we think of r as growing like $\sqrt{n}$. In this case we are essentially keeping the error bound constant. Indeed, in our running time analysis we will assume that λ stays constant as n grows.

We are therefore assuming that Q is a locally distorted similarity transformation of P, with extra and/or missing points. Our problem is to recover the similarity transformation from the two sets of points. In this paper we present a new technique for solving a point pattern matching problem of this nature.

3. THE ALGORITHM

The idea of the algorithm is that, with high probability, one of the first few random points in P will correspond to some point in Q. So we take a random point in P and then search for a point in Q such that it matches the point in P "locally". By that we mean that there exists a similarity transformation that maps the nearest neighbors of P to those of Q. If we find such a map, it is easy to check whether it also gives a "global" match.

We now describe the algorithm more formally.

[1] n is the number of points in the pattern to be matched

3.1. **The main loop.**

input two point sets P and Q and the parameters k, ρ and t.
Step 1. Precomputations
a) For each point in P find the k nearest neighbors (in order of closeness) and store in nearest neighbor list. Do the same for Q. This can be done using the algorithms described in [5] or [4].
b) Divide the smallest square into which Q fits into a two dimensional array of squares of side length $r/\sqrt{n}$. Let each entry in the array contain a list of the points in Q that lie in that square. This will allow us to quickly check whether there is a point in Q at a given coordinate. See section 3.2
Step 2.
until a global match is found, for each i $(1 \leq i \leq n)$ and j $(1 \leq j \leq m)$
Determine whether the k nearest neighbors of p_i matches the k nearest neighbors of q_j. Let T be the similarity transformation that gave the local match. Check whether T can be improved to give a global match. See section 3.3.
Step 3.
output T (that is s, θ, t_x and t_y) and the matching pairs.

3.2. **Finding points in Q.** The precomputation done in Step 1b) above allows one to check, in $O(1)$ time, whether there is a point in Q within a short distance t_0 of any coordinate $q_0(x, y)$. By short we mean $t_0 < r/\sqrt{n}$. We simply find the square (j, k) of the look-up array in which the coordinate falls and then for each of the points, q', occurring in the 8 squares around (j, k) and the square (j, k) check for a match, that is, whether $|q_0 - q'| < t_0$. See figure 1.

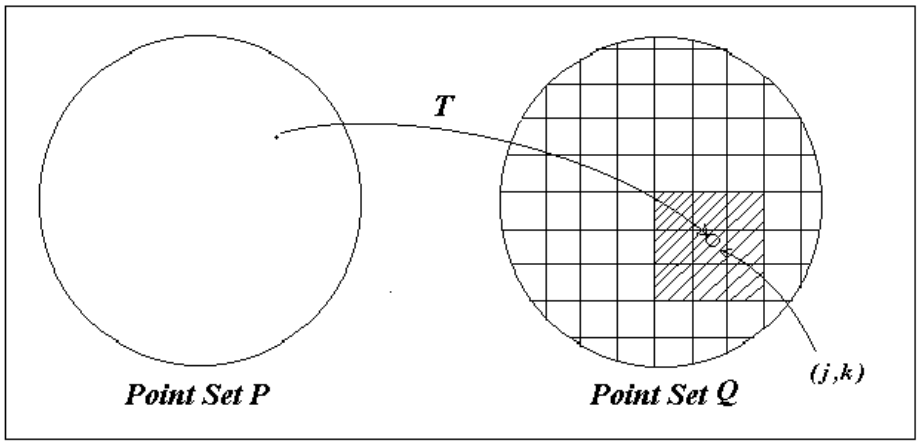

FIGURE 1. Finding points in Q.

3.3. **Comparing nearest neighbors.** This algorithm will depend on two parameters, k_2 and k_3. Guidelines for the best values to choose will be found in the computational complexity section (Section 4), and some particular examples of good choices in the implementation section, Section 5. The idea for checking whether the nearest neighbors of $p \in P$ match those of $q \in Q$, is to assume that two nearest neighbors that are as far away from p and q respectively, as possible, correspond under a similarity transformation. We compute such a similarity transformation and then check

whether enough of the nearest neighbors match under this transformation. More precisely, assume that $\{a_1, a_2, \ldots, a_k\}$ are the k sorted nearest neighbors of the point p in P (with a_1 the nearest neighbor, etc.) and $\{b_1, b_2, \ldots, b_k\}$ the k sorted nearest neighbors of q in Q. Then for each of the k_2 points $a_{k-[k_3/2]-k_2+1}$ to $a_{k-[k_3/2]}$ we compute the similarity transform T sending p to q and $a_{k-[k_3/2]-i}$, in turn, to each of the k_3 b's closest to $b_{k-[k_3/2]-i}$ (that is b_{k-k_3+1-i} to b_{k-i}). For each of these k_2k_3 T's we check whether they give more than $\rho(k-1)$ further nearest neighbor matches. For each T that does, we check whether that T can be refined to give a global match.

input Let $\{a_1, a_2, \ldots, a_k\}$ be the k nearest neighbors of the point p in P.
Let $\{b_1, b_2, \ldots, b_k\}$ be the k nearest neighbors of q in Q.
for $i = 0$ to $k_2 - 1$
  for $j = 0$ to $k_3 - 1$
    let $L_1 = \{p\}$ and $L_2 = \{q\}$.
    let T be the unique transformation that sends p to q and $a_{k-i-[k_3/2]}$ to b_{k-i-j}. Such a T is given by equation 2 below.
    for $l = 1$ to k
      if there is a point, q', in Q within t of $T(a_l)$ (see section 3.2) **then** append a_l to L_1 and append q' to L_2.
    if (the number of points in L_1) -2 is greater than $\rho(k-1)$, then T gives a local match.
      if, using the algorithm in section 3.5 applied to L_1 and L_2, this local match gives a global match
        return the transformation T giving the global match and the matching points.

See figure 2 for an illustration of this algorithm.

Let $p = (p_x, p_y)$ and $a = (a_x, a_y)$ be two distinct points in P, and $q = (q_x, q_y)$ and $b = (b_x, b_y)$ be two distinct points in Q. To find the unique similarity transformation, T, such that $T(p) = q$ and $T(a) = b$, we compute s, θ, t_x and t_y as follows:

$$
\begin{aligned}
s &= \frac{|\overrightarrow{qb}|}{|\overrightarrow{pa}|} \\
\theta &= \text{ angle from } \overrightarrow{pa} \text{ to } \overrightarrow{qb} \\
t_x &= q_x - p_x s\cos(\theta) + p_y s\sin(\theta) \\
t_y &= q_y - p_x s\sin(\theta) - p_y s\cos(\theta).
\end{aligned} \tag{2}
$$

As a practical improvement we might relax the condition "**if** there is a point, q', in Q within t of $T(a_l)$" to the point q' being within a larger multiple of t of $T(a_l)$. In practice we used $|T(a_l) - q'| < 2t$. The reason for this is that T is computed from only two points and therefore even if all three image points q, b_{k-i-j} and q' are within t of their correct positions $T(a_l)$ might be further than t from q'. Note that this makes little difference to our complexity analysis. The only change is in the probability in equation 6, but this only changes the constants implied in our O-notations.

3.4. **Computing a similarity transformation.** In this section we describe an algorithm for computing the "best" similarity transformation mapping a_i to b_i for $i = 1, 2, \ldots, l$, where the a_i are l points in P and b_i are l points in Q.

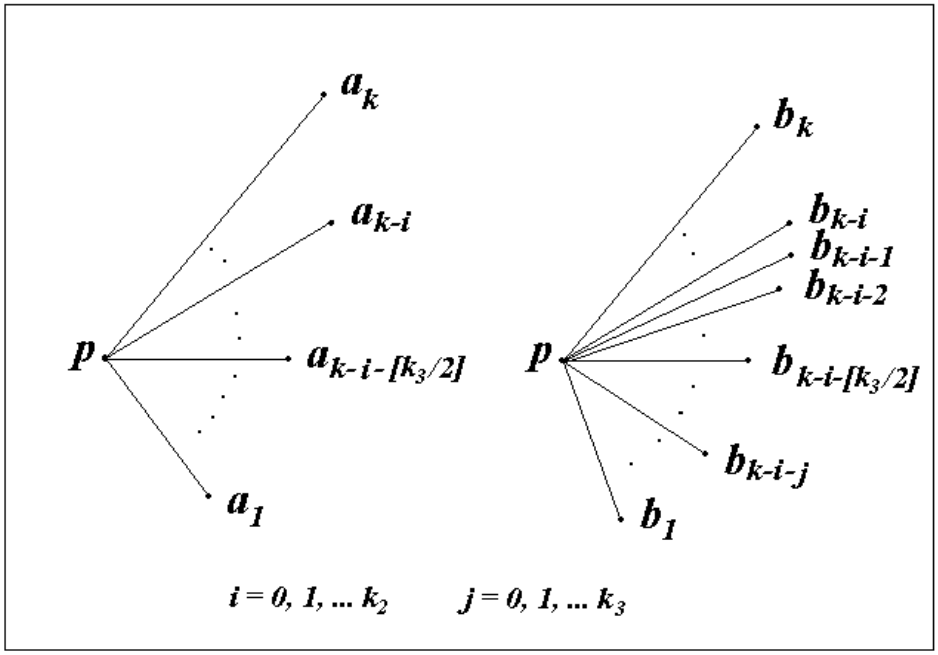

FIGURE 2. The nearest neighbor comparisons.

Let

$$\mu_{x_A} = \sum_{i=1}^{l} x_{a_i}, \qquad \mu_{x_B} = \sum_{i=1}^{l} x_{b_i}$$

$$\mu_{y_A} = \sum_{i=1}^{l} y_{a_i}, \qquad \mu_{y_B} = \sum_{i=1}^{l} y_{b_i}$$

$$l_{A+B} = \sum_{i=1}^{l} (x_{a_i} x_{b_i} + y_{a_i} y_{b_i}), \qquad l_{A-B} = \sum_{i=1}^{l} (x_{a_i} y_{b_i} - y_{a_i} x_{b_i})$$

$$l_A = \sum_{i=1}^{l} (x_{a_i}^2 + y_{a_i}^2)$$

$$D = l l_A - \mu_{x_A}^2 - \mu_{y_A}^2.$$

Then let

$$\overline{r} = \frac{1}{D} \begin{pmatrix} l_A & 0 & -\mu_{x_A} & \mu_{y_A} \\ 0 & l_A & -\mu_{y_A} & -\mu_{x_A} \\ -\mu_{x_A} & -\mu_{y_A} & l & 0 \\ \mu_{y_A} & -\mu_{x_A} & 0 & l \end{pmatrix} \begin{pmatrix} \mu_{x_B} \\ \mu_{y_B} \\ l_{A+B} \\ l_{A-B} \end{pmatrix}.$$

Then $\overline{r} = (t_x, t_y, s\cos\theta, s\sin\theta)^t$ where (s, θ, t_x, t_y) is the similarity transformation that gives the best least-squares match between the $T(a_i)$ and b_i, that is, minimizes $\sum_{i=1}^{l} |T(a_i) - b_i|^2$. For the proof of this fact see [3].

3.5. **Finding a global match.** Assume that we have a list of points $\{a_1, a_2, \ldots, a_l\}$ in P and a list $\{b_1, b_2, \ldots, b_l\}$ in Q such that a_i should match b_i. We describe an algorithm for expanding such a local match to a global match. The algorithm returns a similarity transformation T that gives the global match (if there is one). We start by refining the local match by computing the best least squares match between the points that already match. We then find all points matching under the refined T and repeat the process. We abort if we stop finding new matching points. We declare a global match if at some point we find more than ρn matches.

input The two lists $L_1 = \{a_1, a_2, \ldots, a_l\}$ and $L_2 = \{b_1, b_2, \ldots, b_l\}$
until the number of matches is greater than ρn.
    let T be the similarity transformation we get from applying the algorithm in section 3.4 to L_1 and L_2.
    let $L_1 = \{\}$ and $L_2 = \{\}$.
    unmark all q in Q.
    for $i = 1$ **to** n.
        if the algorithm in section 3.2 finds a point within t of $T(p_i)$ in Q
            let $q \in Q$ be the one closest to $T(p_i)$.
            if q is not marked
                append p_i to L_1 and append q to L_2.
                mark q in Q.
    if L_1 is not longer than it was in the previous iteration
        abort, no global match found.
output T and the matching pairs.

4. Computational Complexity

4.1. **Note on the expected distance to the k'th nearest neighbor.** In [9] it is proved that if we place n points randomly, with uniform distribution, in a disk of radius r then the expected distance from a given point to it's k'th nearest neighbor is given by

$$\frac{r}{\sqrt{\pi}} \frac{\Gamma(k+1/2)}{\Gamma(k)} \frac{1}{\sqrt{n}} \left(1 - \frac{3}{8n} + O\left(\frac{1}{n^2}\right)\right). \tag{3}$$

In particular the expected distance to the nearest neighbor is

$$\frac{r}{2\sqrt{n}} + O\left(\frac{1}{n^{3/2}}\right). \tag{4}$$

Stirling's formula implies that

$$\lim_{k\to\infty} \frac{\Gamma(k+1/2)}{\Gamma(k)\sqrt{k}} = 1, \tag{5}$$

In particular

$$\frac{\Gamma(k+1/2)}{\Gamma(k)} = O(\sqrt{k}),$$

and a good approximation for the distance to the k'th nearest neighbor is therefore

$$\frac{r}{\sqrt{\pi}} \sqrt{\frac{k}{n}}.$$

4.2. **Expected running time.** We assume that n is approximately equal to m. We will study the running time of the algorithm as n goes to infinity but ρ and λ stays constant. In particular, as noted in the introduction, we are assuming that r is growing like $\sqrt{n}$.

The precomputation for finding the nearest neighbors is $O(n \log n + nk \log k)$. See [5].

Constructing the look-up table is clearly $O(n)$.

Suppose that the probability of finding a local match between the k nearest neighbors of a random point $p_i \in P$ and the k nearest neighbors of one of the points in Q, is ρ_0. Then the probability that we will check through l points in P in the main loop without getting a local match is $(1-\rho_0)^l$. So even for $\rho_0 = 0.6$, we should get a local match in 99% of cases after only 5 or less points in P ($0.4^5 = 0.01024$).

In fact we can compute the average number of points we will need to check in P until we find a local match in Q. With probability ρ_0 we will need 1 point. With probability $(1-\rho_0)\rho_0$ we will need 2 points. With probability $(1-\rho_0)^{i-1}\rho_0$ we will need i points. So we will, on average, need

$$\rho_0 \sum_{i=1}^{\infty} (1-\rho_0)^{i-1} i = \rho_0 \frac{1}{\rho_0^2} = \frac{1}{\rho_0}$$

points.

So we see that, assuming there is a match, we will exit the **until** loop over P and Q in Step 2 (Section 3.1) after, on average, n/ρ_0 executions of the body of the **until** loop.

What if there is no match? Assume that we have two point sets and we know that they either match with fixed parameters n, ρ and λ or that they do not match. Such a situation might for instance occur if we have a fingerprint and a large database of fingerprints to match against. In such a situation we can find a good value for ρ_0 by running the algorithm many times on data for which a match *does* exist and computing the average number of points in P that is checked before a correct local match is found (see table 2). By the argument above we know that this value is $1/\rho_0$. Now, for any two point sets we can decide whether they match or not, as follows. Suppose we want the answer to be correct with probability x (say 0.99). Then find l such that $1-(1-\rho_0)^l > x$. Check the first l points in P for a local match. If this gives a global match then, of course, we know(with probability 1) that a match does exist. If, on the other hand, we do not find a match during the first l points, the probability of this happening (assuming there is a match) is very small, $(1-\rho_0)^l < 1-x$. So if we don't find a match within l points, we can say, with probability $1-(1-\rho_0)^l > x$ that in fact none exists.

Note that the probability ρ_0 is very close to, but less than, the ρ of the problem statement. This is because even if p does have a match in Q, our algorithm for finding local matches is not guaranteed to find it. The probability of missing a correct local match will depend on our choice of k, k_2, and k_3. In some cases it might even speed up the overall algorithm by making choices for these parameters that make it more likely that we will miss a correct local match.

Note that checking whether a given similarity transformation T gives a global match takes time at least $O(n)$. This means that we need to make sure that we don't find too many incorrect local matches. So let's see how big we need to choose k in order to find only $O(1)$ incorrect local matches while checking all the points in Q against a particular point in P.

First note that the probability of finding a point within a distance t of a particular random position in Q is

$$n\frac{\pi t^2}{\pi r^2} = n\frac{4\pi\lambda^2 r^2}{\pi n r^2} = \frac{\lambda^2}{4}. \tag{6}$$

Suppose we pick a nearest neighbor of each of $p \in P$ and $q \in Q$ and compute a similarity transformation, T, matching them. We then check whether there is a point in Q within t of each of the k images $T(a_i)$, where the a_i are the nearest neighbors of p. If we find ρk matches or more, we declare a success. What is the probability of an incorrect T passing this test? If we assume that each of the k tests are independent and using the probability found in the previous paragraph, we see that the probability of finding at least ρk matches in k tests is

$$\sum_{i=0}^{\rho k} \binom{k}{i} \left(\frac{\lambda^2}{4}\right)^i \left(1 - \frac{\lambda^2}{4}\right)^{k-i}.$$

It is well known, see [6] or [8], that this tail of the binomial distribution is bounded by

$$e^{-2(\rho - \lambda^2/4)^2 k}.$$

Therefore we will choose k to be bigger than

$$\frac{\ln n}{2(\rho - \lambda^2/4)^2}.$$

This ensures that we will get only $O(1)$ incorrect local matches in every n runs.

We need to choose k_2 and k_3 large enough such that we do not miss a local match if there is one. The problem is that because of the noise in the data the images of p's nearest neighbors in Q might occur in a different order. Also the missing/extra points forces us to look at extra neighbors. By the same argument as for the number of points in P we will need to consider, we see that after checking k_2 of the nearest neighbors of p we will have found a point that does occur in Q with probability $1 - (1 - \rho)^{k_2}$. We want this probability to be, say, 95%. This probability need not depend on the size of n because we only need this test to succeed once. We therefore choose k_2 approximately equal to $\ln 0.05 / \ln(1 - \rho)$, in particular k_2 is $O(1)$. On the other hand the nearest neighbors changing their order is a more serious problem. By Equation 3, the expected distance to the k'th nearest neighbors of $q \in Q$ is about

$$r_k = \frac{r}{\sqrt{\pi}} \frac{\Gamma(k + 1/2)}{\Gamma(k)} \frac{1}{\sqrt{n}}.$$

A ring with outer radius $r_k + t$ and inner radius $r_k - t$ has area $4\pi r_k t$, and so we can expect there to be

$$n \frac{4\pi r_k t}{\pi r^2} = \frac{2\lambda}{\sqrt{\pi}} \frac{\Gamma(k+1)}{\Gamma(k)},$$

points in that ring. This means that the images of two consecutive nearest neighbors of p could be $O(\sqrt{k})$ apart in the sorted nearest neighbor list of q. We therefore choose k_3 approximately equal to $2\lambda\sqrt{k}/\sqrt{\pi}$ (see Equation 5).

Given T it takes time $O(k)$ to count the number of local matches. Thus we see that the local matching algorithm in section 3.3 takes time $O(kk_2k_3) = O(k^{3/2})$.

How long does it take to go from a correct local match to a global match? Note that if we computed a T from a set of points in a disk of radius r_0 then the accuracy of s and θ (more so than that of t_x and t_y) will determine in how big a disk this T will now find matching points. Let's assume the correct T is T_0 and we have a T' with the same t_x and t_y, but slightly different θ and s, then the difference between $T_0(p)$ and $T'(p)$ is proportional to $|p|$ (we are assuming that the center of the set P is close to the origin). Conversely, the improvement in T we get from "averaging" over the points in a disk of radius r_0 (using the algorithm in section 3.4) is also (at least) proportional to r_0. This is for essentially the same reasons. The further a point is from the center the stronger it's

influence on the accuracy of s and θ during the least squares fitting. (In fact there are $O(r_0)$ points close to the perimeter of the disk in which we are fitting points and it might therefore be argued that the improvement is better than linear in r_0.) We therefore see that there is some constant α such that if we recompute T by using the matching points in a disk of radius r_0 then we can expect the new T to give matching points in a disk of radius αr_0. So to find the global match we will repeat the "until" loop in section 3.5 only $O(\ln n)$ times. The total time for finding a global match from a local one is therefore $O(n \ln n)$.

Recall that we execute the inner loop in the algorithm of section 3.3 a total of $nk_2k_3 = O(n(\log n)^{1/2})$ times, and that the probability that any one of them will succeed is $O(1/n)$ (by our choice of k). We therefore expect to have to check $O((\log n)^{1/2})$ local matches to see if they become global matches. Each such test takes time $O(n \log n)$. So this part of the algorithm takes time $O(n(\log n)^{3/2})$. The tests for local matches is also time $O(n(\log n)^{3/2})$ and the precomputations are $O(n \log n \log \log n)$ so in total our algorithm has expected running time

$$O(n(\log n)^{3/2}).$$

5. Implementation

The algorithm described above was implemented in the C programming language on a 400MHz Sun 450 with 640MB of RAM. We did a case study on an actual fingerprint and tested the program on many randomly generated data sets.

5.1. **Case study.** To test our algorithm on a real world situation, we applied it to a fingerprint recognition problem. We took two fingerprint images from the same person and extracted the feature points. See figure 3. This gives two point sets, each with 40 points, that we tried to match using our implementation. The program easily found a match involving 32 points. If we relax the matching distance up to 4 more points can be matched. In figure 4 we give a representation of the match found. The "x" marks represent the points in the set Q. The dots represent the images of the points in the P set (under the similarity transformation found by the program). Each point that was matched is circled with a circle of radius t, where t is the matching distance that was given to the program.

FIGURE 3. Two different fingerprints from the same finger with the feature points circled.

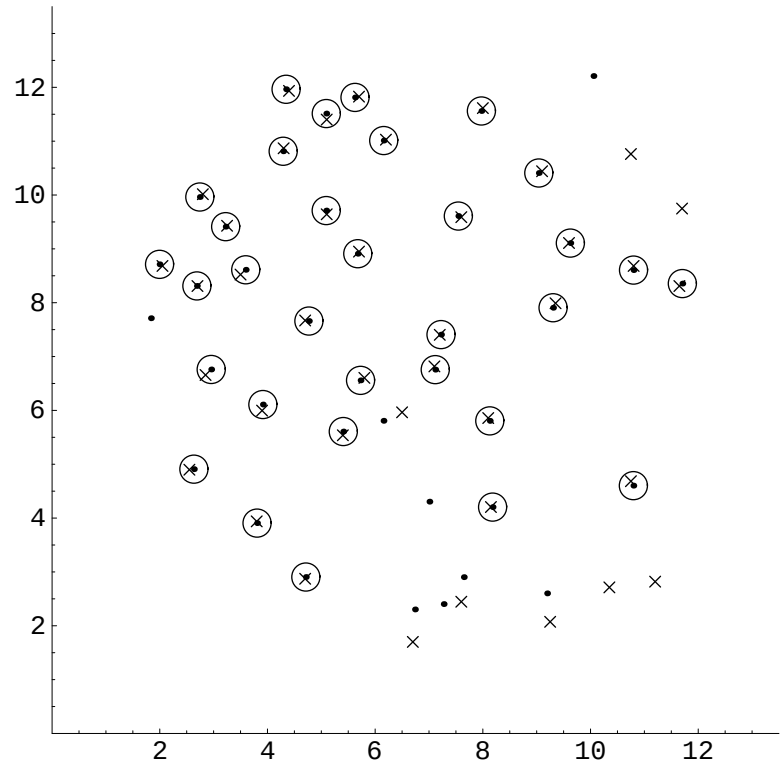

FIGURE 4. The match our implementation found between the two sets of fingerprint feature points.

5.2. **Random point sets.** We also tested the program on a large number of randomly generated point sets. Some of the results are reported in table 2. For each row of the table, we picked values for n, ρ and λ and then tried to find the values for k, k_2, k_3 and k_0 (the meaning of k_0 will be explained shortly) that gave a 100% success rate (out of 100 trials) and minimized the average time to find the match. So the values for the k's reported in Table 2 are close to optimal for the given n, ρ and λ. In the algorithm as described above, we declare a local match if $\rho(k-1)$ of the nearest neighbors match under a certain similarity transformation (see section 3.3). When k is small we might want to modify this slightly, and so, for these tests, we declared a local match if k_0 or more nearest neighbors matched. In this way the running times can sometimes be improved over just choosing $k_0 = [\rho(k-1)]$.

For each set of values n, ρ, λ, k, k_2, k_3 and k_0, 100 trials were run and the average CPU time computed. For each trial two point patterns were generated as follows. The point set P was generated by randomly (with uniform distribution) picking n points in a disk of radius 1 around the origin. A random similarity transformation, T, was then applied to the first $[\rho n]$ of these points. A further $n-[\rho n]$ random points were found in the unit disk and T was applied to them. Noise was then added to these n points in order to make up the set Q. To add noise to a point, each point was moved to a random position (again with uniform distribution) in the disk of radius $t = \lambda r/(2\sqrt{n})$ centered at the original position of the point.

For each 100 runs we report the following averages. The n_p column is the average number of points in the set P that was checked before a correct local match was found. The n_i column is the average number of incorrect local matches that was found for each point in P. Note that this should be divided by k_2k_3 in order to give an accurate idea of how likely an incorrect local match

is for a given k. The τ column is the CPU time, in seconds, used by the matching part of the algorithm (the precomputation time excluded, this turns out to be small compared to the matching time anyway). Finally the c column is the average number of distances between points that were computed. This gives a good machine independent measure of how long the program takes for different choices of the parameters.

As can be seen from the table (and a little computation) the best values for k, k_2 and k_3 are very close to those predicted by the theoretical arguments in section 4, that is

$$k \approx \frac{\ln n}{2(\rho - \lambda^2/4)^2}$$
$$k_2 \approx \frac{\ln 0.05}{\ln(1-\rho)}$$
$$k_3 \approx \frac{2\lambda\sqrt{k}}{\sqrt{\pi}}.$$

TABLE 2. For each set of arguments, the best parameters and the average results over 100 trials. Here n is the number of points, ρ is the matching probability, λ is the matching factor and τ is the best average running time in seconds.

arguments			parameters				results			
n	ρ	λ	k	k_2	k_3	k_0	n_p	n_i	τ	c
50	0.9	0.4	5	1	2	2	3.29	2.40	0.020	15937
100	0.9	0.4	5	1	2	3	3.37	0.44	0.042	36980
200	0.9	0.4	6	1	2	3	3.11	1.63	0.099	89297
500	0.9	0.4	7	1	2	4	3.96	0.53	0.334	316118
1000	0.9	0.4	8	1	2	5	4.53	0.19	0.894	866492
50	0.6	0.5	8	3	3	3	4.21	11.39	0.144	110629
100	0.6	0.5	9	2	4	4	4.63	8.25	0.372	335933
200	0.6	0.5	12	3	4	5	3.28	16.50	0.903	813338
50	0.9	0.9	7	1	3	5	5.64	3.73	0.056	43346
100	0.9	0.9	7	1	3	5	6.71	8.06	0.177	157853
200	0.9	0.9	8	1	5	5	3.70	34.43	0.456	398546
500	0.9	0.9	8	1	4	6	8.54	24.78	1.992	1813054

In the table we present some of the extremes that our algorithm can handle. The first 5 entries is the ideal situation where λ is small, meaning that the positions of the points are relatively accurate and ρ is high: in this case 90% of points can be matched. As can be seen from the table the running time increases roughly linearly with the number of points.

The next 3 entries represent cases where only 60% of points can be matched and the error bound is relatively big, one half the average distance to the nearest neighbor. Even in these cases the algorithm succeeds in finding matches. This value, 0.6, is about as low as the algorithm will tolerate. For lower values of ρ the success rate of the algorithm starts dropping below 100%. This is not too surprising, after all it becomes unclear whether a match would even always exist for so many missing/extra points.

The last 4 entries represent the case where the error bound is now very big: 90% of the average distance to the nearest neighbor. Again the algorithm succeeds in finding the match (note that ρ is relatively large though).

The behavior of the algorithm for intermediate values for ρ and λ can be extrapolated from the table.

Note that n_p is the number that was estimated to be $1/\rho_0$ in the beginning of Section 4.2. As can be seen from the table n_p is indeed bigger when ρ is smaller.

See figure 5 for examples of what each of the three types of matches for 50 points reported in table 2, look like.

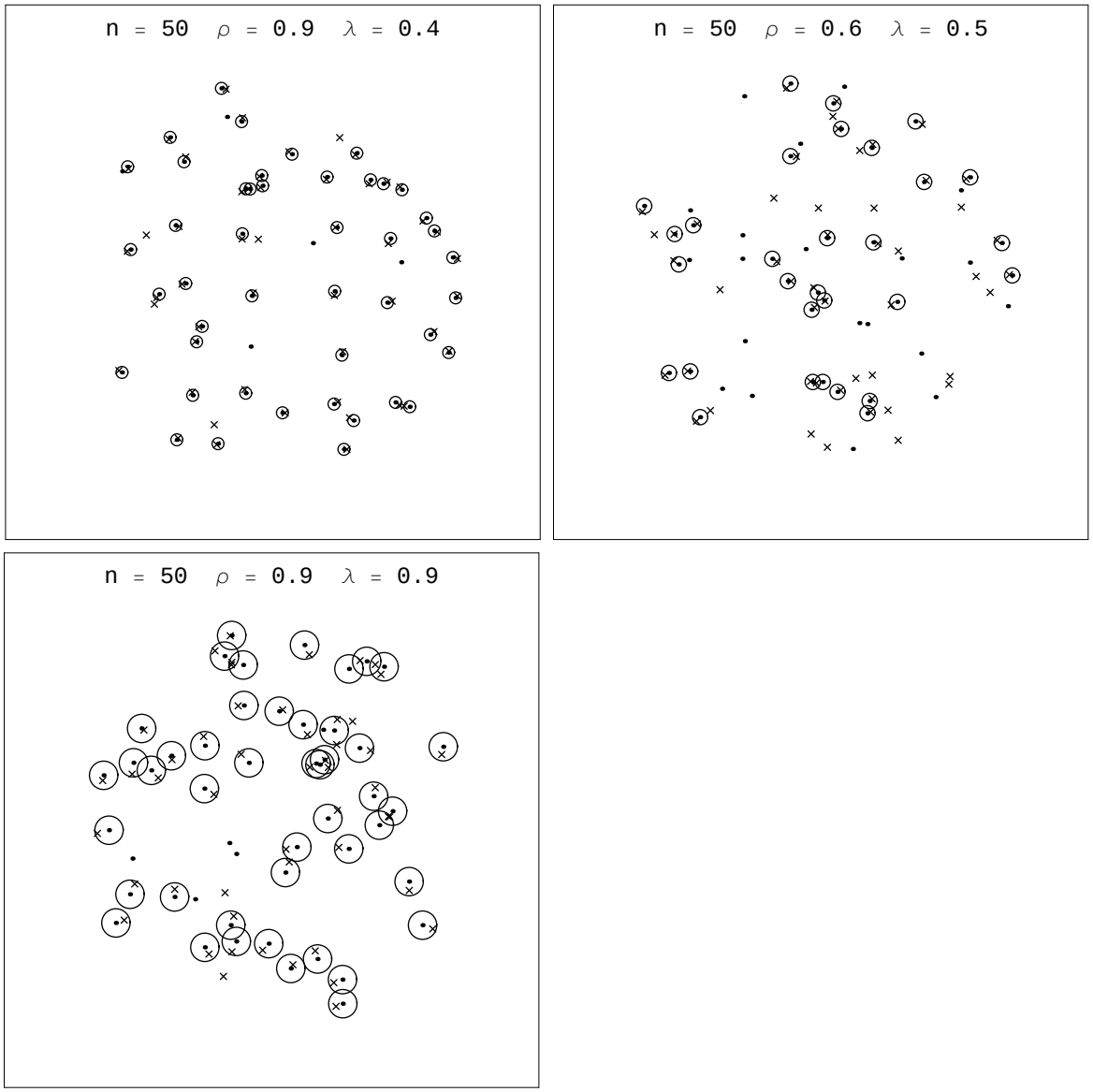

FIGURE 5. Examples of matches found for different values of the arguments n, ρ and λ.

5.3. **Non uniformly distributed data sets.** We will discuss the behavior of the algorithm in case the data sets have outliers or clutter. Note that when we discuss outliers and cluttering, we are of course violating our assumption of uniform distribution and our choice for t (Formula 1) would not be good. So for this discussion we will assume that t was chosen using our knowledge

of the data set we are working on. For instance, if we are dealing with data for which the only uncertainty in the positions of points are due to errors in measurements we probably have a good estimate for the size of such errors, and of course we choose t accordingly.

5.3.1. *Outliers.* If our point sets both contain a few outliers that do match up this could actually help speed up the algorithm. For if we get lucky and one of the outliers is one of the first points in P that we try to match locally to a point in Q, the local match will be very good. This is true because the distances to the nearest neighbors of an outlier are large and therefore the transformation we compute from them will be more accurate. Of course if we start the search with a normal point of P, then the algorithm will behave as usual and we will find the global match as usual. Note that in this case we might miss the match between two outliers if they are very far away from the other points. This is because even a least squares match between the regular matching points might not give an accurate enough transformation to put the image of an outlier in P within t of the outlier in Q.

5.3.2. *Clutter.* In case of a lot of clustering of points the average distance to the nearest neighbors of points in our set will be smaller than that expected of uniformly distributed points. This implies that we would need to increase the size of k in order to get accurate enough local matches so that our algorithm for improving a local match to a global one can be expected to succeed. Increasing k slows down the algorithm and we therefore see that our algorithm will behave poorly in the presence of cluttering. In extreme cases the following might even happen: Every transformation giving a least squares match between points of a cluster in P with the correct cluster in Q is not good enough to match any other points. In this case our algorithm will completely fail to give a global match. To make the algorithm work in this case we will need to make k bigger than the number of points in the average cluster.

6. Conclusion

The design and analysis of data structures and algorithms is an important area of point pattern matching algorithms. More importantly, analyzing point set pattern matching is an integral component of pattern recognition problems. The goal of much of the processing is to group, or organize the number of points in terms of common feature properties.

The intent of this paper is the design and analysis of a probabilistic similarity transformation matching algorithm.

If n is the number of points in the patterns to be matched, we give a $O(n(\log n)^{3/2})$ expected time algorithm for the point pattern matching problem and show that it is faster than any existing algorithms in the literature. We then describe some experimental results on both fingerprints and randomly generated data for the validation of our theoretical analysis. These results show significant improvements in running time. Our theoretical analysis is heuristic in nature but seems to agree well with the experimental results.

Our experimental results show that our algorithm is applicable to a wide variety of problems. The algorithm performs well even if the allowed error is as big as 90% of the average shortest distance to the nearest neighbor or the number of missing/extra points is high: even with only 60% of points matching the algorithm will succeed. Although we have concentrated on uniformly distributed point sets, there is good reason to believe that the algorithm will also work on data sets with outliers or clustering.

References

[1] L. Boxer, Faster point set pattern matching in 3-D. *Pattern Recognition Letters*, **19** (1998) 1235–1240.

[2] S.-H. Chang, F.-H. Cheng, W.-H. Hsu. An $O(n^2)$ Algorithm for 2-D Point Pattern Matching. Submitted to *Pattern Recognition Letters.*

[3] S.-H. Chang, F.-H. Cheng, W.-H. Hsu, G.-Z. Wu. Fast algorithm for point pattern matching: Invariant to translations, rotations and scale changes. *Pattern Recognition*, 1997 30(2):311–320.

[4] M. T. Dickerson,R. L. Drysdale, and J-R Sack. Simple algorithms for enumerating inter-point distances and finding k nearest neighbors. *Internat. J. Comput. Geom. Appl.*, 1992 **2**(3):221–239.

[5] M.T. Dickerson, D. Eppstein. Algorithms for proximity problems in higher dimensions. *Computational Geometry*, 1996 5:277–291.

[6] N. L. Johnson, S. Kotz. *Distributions in statistics: Discrete distributions.* Houghton Mifflin Co., Boston, Mass. 1969.

[7] H. Ogawa. Labeled point pattern matching by fuzzy relaxation. *Pattern Recognition*, **17**(5) (1984): 569–573.

[8] M. Okamoto. Some inequalities relating to the partial sum of binomial probabilities. *Annals of the Institute of Statistical Mathematics, Tokyo* **10** (1958) 29–35.

[9] A. G. Percus, O. C. Martin Scaling universalities of kth-nearest neighbor distances on closed manifolds *Adv. in Appl. Math.*, **21** (1998) 3, 424–436

[10] S. Ranade, A. Rosenfeld. Point pattern matching by relaxation. *Pattern Recognition*, **12** (1980) 269–275.

[11] V. V. Vinod, S. Ghose. Point matching using asymmetric neural networks. *Pattern Recognition*, **26**(8) (1993) (1207–1214.

Department of Mathematics, Louisiana State University, Baton Rouge, LA 70803, USA
E-mail address: wamelen@math.lsu.edu

Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA
E-mail address: zhili@bit.csc.lsu.edu

Department of Computer Science, Louisiana State University, Baton Rouge, LA 70803, USA
E-mail address: iyengar@bit.csc.lsu.edu